A Non-linear Arithmetic Procedure for Control-Command Software Verification

Pierre Roux, Mohamed Iguernelala, Sylvain Conchon

A Non-linear Arithmetic Procedure for Control-Command Software Verification*

Pierre Roux[1], Mohamed Iguernlala[2,3], and Sylvain Conchon[3,4]

[1] ONERA, DTIS, Toulouse F-31055
[2] OCamlPro SAS, Gif-sur-Yvette F-91190
[3] LRI, Université Paris-Sud, Orsay F-91405
[4] INRIA Saclay – Ile-de-France, Toccata, Orsay F-91893

Abstract. State-of-the-art (semi-)decision procedures for non-linear real arithmetic address polynomial inequalities by mean of symbolic methods, such as quantifier elimination, or numerical approaches such as interval arithmetic. Although (some of) these methods offer nice completeness properties, their high complexity remains a limit, despite the impressive efficiency of modern implementations. This appears to be an obstacle to the use of SMT solvers when verifying, for instance, functional properties of control-command programs.
Using off-the-shelf convex optimization solvers is known to constitute an appealing alternative. However, these solvers only deliver approximate solutions, which means they do not readily provide the soundness expected for applications such as software verification. We thus investigate a-posteriori validation methods and their integration in the SMT framework. Although our early prototype, implemented in the Alt-Ergo SMT solver, often does not prove competitive with state of the art solvers, it already gives some interesting results, particularly on control-command programs.

Keywords: SMT, non-linear real arithmetic, polynomial inequalities, convex optimization

1 Introduction

Systems of non-linear polynomial constraints over the reals are known to be solvable since Tarski proved that the first-order theory of the real numbers is decidable, by providing a quantifier elimination procedure. This procedure has then been much improved, particularly with the cylindrical algebraic decomposition. Unfortunately, its doubly exponential complexity remains a serious limit to its scalability. It is now integrated into SMT solvers [23]. Although it demonstrates very good practical results, symbolic quantifier elimination seems to remain an obstacle to scalability on some problems. In some cases, branch and bound with interval arithmetic constitutes an interesting alternative [17].

* This work has been partially supported by the French ANR projects ANR-12-INSE-0007 Cafein and ANR-14-CE28-0020 Soprano and the project SEFA IKKY.

We investigate the use of numerical optimization techniques, called semi-definite programming, as an alternative. We show in this paper how solvers based on these techniques can be used to design a sound semi-decision procedure that outperforms symbolic and interval-arithmetic methods on problems of practical interest. A noticeable characteristic of the algorithms implemented in these solvers is to only compute approximate solutions.

We explain this by making a comparison with linear programming. There are two competitive methods to optimize a linear objective under linear constraints: the interior point and the simplex algorithms. The interior point algorithm starts from some initial point and performs steps towards an optimal value. These iterations converge to the optimum but not in finitely many steps and have to be stopped at some point, yielding an approximate answer. In contrast, the simplex algorithm exploits the fact that the feasible set is a polyhedra and that the optimum is achieved on one of its vertices. The number of vertices being finite, the optimum can be exactly reached after finitely many iterations. Unfortunately, this nice property does not hold for spectrahedra, the equivalent of polyhedra for semi-definite programming. Thus, all semi-definite programming solvers are based on the interior-point algorithm, or a variant thereof.

To illustrate the consequences of these approximate solutions, consider the proof of $e \leq c$ with e a complicated ground expression and c a constant. $e \leq c$ can be proved by exactly computing e, giving a constant c', and checking that $c' \leq c$. However, if e is only approximately computed: $e \in [c' - \epsilon, c' + \epsilon]$, this is conclusive only when $c' + \epsilon \leq c$. In particular, if e is equal to c, an exact computation is required. This inability to prove inequalities that are not satisfied with some margin is a well known property of numerical verification methods [42] which can then be seen as a trade-off between completeness and computation cost.

The main point of this paper is that, despite their incompleteness, numerical verification methods remain an interesting option when they enable to practically solve problems for which other methods offer an untractable complexity. Our contributions are:

(1) a comparison of two sound semi-decision procedures for systems of non-linear constraints, which rely on off-the-shelf numerical optimization solvers,
(2) an integration of these procedures in the Alt-Ergo SMT solver,
(3) an experimental evaluation of our approach on a set of benchmarks coming from various application domains.

The rest of this paper is organized as follows: Section 2 gives a practical example of a polynomial problem, coming from control-command program verification, better handled by numerical methods. Section 3 is dedicated to preliminaries. It introduces basic concepts of sum of squares polynomials and semi-definite programming. In Section 4, we compare two methods to derive sound solutions to polynomial problems from approximate answers of semi-definite programming solvers. Section 5 provides some implementation details and discuss experimental results. Finally, Section 6 concludes with some related and future works.

```
typedef struct { double x0, x1, x2; } state;
/*@ predicate inv(state *s) = 6.04 * s->x0 * s->x0 - 9.65 * s->x0 * s->x1
  @   - 2.26 * s->x0 * s->x2 + 11.36 * s->x1 * s->x1
  @   + 2.67 * s->x1 * s->x2 + 3.76 * s->x2 * s->x2 <= 1; */

/*@ requires \valid(s) && inv(s) && -1 <= in0 <= 1;
  @ ensures inv(s); */
void step(state *s, double in0) {
  double pre_x0 = s->x0, pre_x1 = s->x1, pre_x2 = s->x2;
  s->x0 = 0.9379 * pre_x0 - 0.0381 * pre_x1 - 0.0414 * pre_x2 + 0.0237 * in0;
  s->x1 = -0.0404 * pre_x0 + 0.968 * pre_x1 - 0.0179 * pre_x2 + 0.0143 * in0;
  s->x2 = 0.0142 * pre_x0 - 0.0197 * pre_x1 + 0.9823 * pre_x2 + 0.0077 * in0;
}
```

Fig. 1: Example of a typical control-command code in C.

2 Example: Control-Command Program Verification

Control-command programs usually iterate linear assignments periodically over time. These assignments take into account a measure (via some *sensor*) of the state of the physical system to control (called *plant* by control theorists) to update an internal state and eventually output orders back to the physical system (through some *actuator*). Figure 1 gives an example of such an update, `in0` being the input and `s` the internal state. The comments beginning by @ in the example are annotations in the ACSL language [12]. They specify that before the execution of the function (`requires`) `s` must be a valid pointer satisfying the predicate `inv` and $|\texttt{in0}| \leq 1$ must hold. Under these hypotheses, `s` still satisfies `inv` after executing the function (`ensures`).

To prove that the internal state remains bounded over any execution of the system, a quadratic polynomial[5] can be used as invariant[6]. Checking the validity of these invariants then leads to arithmetic verification conditions (VCs) involving quadratic polynomials. Such VCs can for instance be generated from the program of Figure 1 by the Frama-C/Why3 program verification toolchain [12,16]. Unfortunately, proving the validity of these VCs seem out of reach for current state-of-the-art SMT solvers. For instance, although Z3 [13] can solve smaller examples with just two internal state variables in a matter of seconds, it ran for a few days on the three internal state variable example of Figure 1 without reaching a conclusion[7]. In contrast, our prototype can prove it in a fraction of second, as well as other examples with up to a dozen variables.

Verification of control-command programs is a good candidate for numerical methods. These systems are designed to be robust to many small errors, which means that the verified properties are usually satisfied with some margin. Thus, the incompleteness of numerical methods is not an issue for this kind of problems.

[5] For instance, the three variables polynomial in *inv* in Figure 1.

[6] Control theorists call these invariants sublevel sets of a quadratic Lyapunov function. Such functions exist for linear systems if and only if they do not diverge.

[7] This is the case even on a simplified version with just arithmetic constructs, i.e., expurgated of all the reasoning about pointers and the C memory model.

3 Preliminaries

3.1 Emptiness of Semi-algebraic Sets

Our goal is to prove that conjunctions of polynomial inequalities are unsatisfiable, that is, given some polynomials with real coefficients $p_1, \ldots, p_m \in \mathbb{R}[x]$, we want to prove that there does not exist any assignment for the n variables $x_1, \ldots, x_n \in \mathbb{R}^n$ such that all inequalities $p_1(x_1, \ldots, x_n) \geq 0, \ldots, p_m(x_1, \ldots, x_n) \geq 0$ hold simultaneously. In the rest of this paper, the notation $p \geq 0$ (resp. $p > 0$) means that for all $x \in \mathbb{R}^n$, $p(x) \geq 0$ (resp. $p(x) > 0$).

Theorem 1. *If there exist polynomials $r_i \in \mathbb{R}[x]$ such that*

$$-\sum_i r_i\, p_i > 0 \quad \textit{and} \quad \forall i, r_i \geq 0 \tag{1}$$

then the conjunction $\bigwedge_i p_i \geq 0$ is unsatisfiable[8].

Proof. Assume there exist $x \in \mathbb{R}^n$ such that for all i, $p_i(x) \geq 0$. Then, since $r_i \geq 0$, we have $r_i(x)\, p_i(x) \geq 0$ hence $\left(\sum_i r_i\, p_i\right)(x) \geq 0$ which contradicts $-\sum_i r_i\, p_i > 0$.

In fact, under some hypotheses[9] on the p_i, the condition (1) is not only sufficient but also necessary, as stated by the Putinar's Positivstellensatz [27, §2.5.1]. Unfortunately, no practical bound is known on the degrees of the polynomials r_i. In our prototype, we restrict the degrees of each r_i to[10] $d - \deg(p_i)$ where $d := \max_i(\deg(p_i))$, so that $\sum_i r_i\, p_i$ is a polynomial of degree d. This is a first source of incompleteness, although benchmarks show that it already enables to solve many interesting problems.

The sum of squares (SOS) technique [26,36] is an efficient way to numerically solve polynomial problems such as (1). The next sections recall its main ideas.

3.2 Sum of Squares (SOS) Polynomials

A polynomial $p \in \mathbb{R}[x]$ is said to be SOS if there exist polynomials $h_i \in \mathbb{R}[x]$ such that for all x,

$$p(x) = \sum_i h_i^2(x).$$

Although not all non negative polynomials are SOS, being SOS is a sufficient condition to be non negative.

Example 1 (from [36]). Considering $p(x_1, x_2) = 2x_1^4 + 2x_1^3x_2 - x_1^2x_2^2 + 5x_2^4$, there exist $h_1(x_1, x_2) = \frac{1}{\sqrt{2}}\left(2x_1^2 - 3x_2^2 + x_1x_2\right)$ and $h_2(x_1, x_2) = \frac{1}{\sqrt{2}}\left(x_2^2 + 3x_1x_2\right)$ such that $p = h_1^2 + h_2^2$. This proves that for all $x_1, x_2 \in \mathbb{R}$, $p(x_1, x_2) \geq 0$.

[8] Or, with different words, the semi-algebraic set $\{x \in \mathbb{R}^n \mid \forall i, p_i(x) \geq 0\}$ is empty.

[9] For instance, when one of the sets $\{x \in \mathbb{R}^n \mid p_i(x) \geq 0\}$ is bounded.

[10] More precisely to $2\left\lceil \frac{d - \deg(p_i)}{2} \right\rceil$ as $deg(r_i)$ is necessarily even since $r_i \geq 0$.

Any polynomial p of degree $2d$ (a non negative polynomial is necessarily of even degree) can be written as a quadratic form in the vector of all monomials of degree less or equal to d:

$$p(x) = z^T Q\, z \tag{2}$$

where $z = \left[1, x_1, \ldots, x_n, x_1x_2, \ldots, x_n^d\right]^T$ and Q is a constant symmetric matrix.

Example 2. For $p(x_1, x_2) = 2x_1^4 + 2x_1^3x_2 - x_1^2x_2^2 + 5x_2^4$, we have[11]

$$\begin{aligned} p(x_1, x_2) &= \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1x_2 \end{bmatrix}^T \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{12} & q_{22} & q_{23} \\ q_{13} & q_{23} & q_{33} \end{bmatrix} \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1x_2 \end{bmatrix} \\ &= q_{11}x_1^4 + 2q_{13}x_1^3x_2 + (q_{33} + 2q_{12})x_1^2x_2^2 + 2q_{23}x_1x_2^3 + q_{22}x_2^4. \end{aligned}$$

Thus $q_{11} = 2$, $2q_{13} = 2$, $q_{33} + 2q_{12} = -1$, $2q_{23} = 0$ and $q_{22} = 5$. Two possible examples for the matrix Q are shown below:

$$Q = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 5 & 0 \\ 1 & 0 & -3 \end{bmatrix}, \qquad Q' = \begin{bmatrix} 2 & -3 & 1 \\ -3 & 5 & 0 \\ 1 & 0 & 5 \end{bmatrix}.$$

The polynomial p is then SOS if and only if there exists a positive semi-definite matrix Q satisfying (2). A matrix Q is called positive semi-definite, noted $Q \succeq 0$, if, for all vector x, $x^T Q\, x \geq 0$. Just as a scalar $q \in \mathbb{R}$ is non negative if and only if $q = r^2$ for some $r \in \mathbb{R}$ (typically $r = \sqrt{q}$), $Q \succeq 0$ if and only if $Q = R^T R$ for some matrix R (then, for all x, $x^T Q x = (Rx)^T(Rx) = \|Rx\|_2^2 \geq 0$). The vector Rz is then a vector of polynomials h_i such that $p = \sum_i h_i^2$.

Example 3. In the previous example, the matrix Q is not positive semi-definite (for $x = [0, 0, 1]^T$, $x^T Q\, x = -3$). In contrast, $Q' \succeq 0$ as $Q' = R^T R$ with

$$R = \frac{1}{\sqrt{2}} \begin{bmatrix} 2 & -3 & 1 \\ 0 & 1 & 3 \end{bmatrix}$$

giving the decomposition of Example 1.

3.3 Semi-Definite Programming (SDP)

Given symmetric matrices $C, A_1, \ldots, A_m \in \mathbb{R}^{s \times s}$ and scalars $a_1, \ldots, a_m \in \mathbb{R}$, the following optimization problem is called *semi-definite programming*

$$\begin{aligned} \text{minimize} \quad & \operatorname{tr}(CQ) \\ \text{subject to} \quad & \operatorname{tr}(A_1Q) = a_1 \\ & \quad \vdots \\ & \operatorname{tr}(A_mQ) = a_m \\ & Q \succeq 0 \end{aligned} \tag{3}$$

where the symmetric matrix $Q \in \mathbb{R}^{s \times s}$ is the variable, $\operatorname{tr}(M) = \sum_i M_{i,i}$ denotes the trace of the matrix M and $Q \succeq 0$ means Q positive semi-definite.

[11] All monomials of p are of degree 4, so z does not need to contain 1, x_1 and x_2.

Remark 1. Since the matrices are symmetric, $\mathrm{tr}(AQ) = \mathrm{tr}(A^T Q) = \sum_{i,j} A_{i,j} Q_{i,j}$. The constraints $\mathrm{tr}(AQ) = a$ are then affine constraints between the entries of Q.

As we have just seen in Section 3.2, existence of a SOS decomposition amounts to existence of a positive semi-definite matrix satisfying a set of affine constraints, that is a solution of a semi-definite program. Semi-definite programming is a convex optimization problem for which there exist efficient numerical solvers [7,44], thus enabling to solve problems involving polynomial inequalities over the reals.

3.4 Parametric Problems

Up to now, we have only seen how to check whether a given polynomial p with fixed coefficients is SOS (which implies its non negativeness). However, according to Section 3.1, we need to solve problems in which polynomials p have coefficients that are not fixed but parameters. One of the great strengths of SOS programming is its ability to solve such problems.

An unknown polynomial $p \in \mathbb{R}[x]$ of degree d with n variables can be written

$$p = \sum_{\alpha_1 + \cdots + \alpha_n \leq d} p_\alpha x_1^{\alpha_1} \dots x_n^{\alpha_n}$$

where the p_α are scalar parameters. A constraint such as $r_i \geq 0$ in (1) can then be replaced by r_i is SOS, that is: $\exists\, Q \succeq 0, r_i = z^T Q\, z$, which is a set of affine equalities between the coefficients of Q and the coefficients $r_{i,\alpha}$ of r_i. This can be cast as a semi-definite programming problem [12].

Thus, problems with unknown polynomials p, as the one presented in Section 3.1, can be numerically solved through SOS programming.

Remark 2 (Complexity). The number s of monomials in n variables of degree less than or equal to d, i.e., the size of the vector z in the decomposition $p(x) = z^T Q\, z$, is $s := \binom{n+d}{d}$. This is polynomial in n for a fixed d (and vice versa). In practice, current SDP solvers can solve problems where s is about a few hundreds. This makes the SOS relaxation tractable for small values of n and d ($n \sim 10$ and $d \sim 3$, for instance). Our benchmarks indicate this is already enough to solve some practical problems that remain out of reach for other methods.

4 Numerical Verification of SOS

According to Section 3.1, a conjunction of polynomial constraints can be proved unsatisfiable by exhibiting other polynomials satisfying some constraints. Section 3.4 shows that such polynomials can be efficiently found by some numerical optimization solvers. Unfortunately, due to the algorithms they implement, we cannot directly trust the results of these solvers. This section details this issue and reviews two a-posteriori validation methods, with their respective weaknesses.

[12] By encoding the $r_{i,\alpha} \in \mathbb{R}$ as $r^+_{i,\alpha} - r^-_{i,\alpha}$ with $r^+_{i,\alpha}, r^-_{i,\alpha} \geq 0$ and putting the new variables in a block diagonal matrix variable $Q' := \mathrm{diag}(Q, \dots, r^+_{i,\alpha}, r^-_{i,\alpha}, \dots)$.

4.1 Approximate Solutions from SDP Solvers

In practice, the matrix Q returned by SDP solvers upon solving an SDP problem (3) does not precisely satisfy the equality constraints, due both to the algorithms used and their implementation with floating-point arithmetic. Therefore, although the SDP solver returns a positive answer for a SOS program, this does not constitute a valid proof that a given polynomial is SOS.

Most SDP solvers start from some $Q \succeq 0$ not satisfying the equality constraints (for instance the identity matrix) and iteratively modify it in order to reduce the distance between $\mathrm{tr}(A_i Q)$ and a_i while keeping Q positive semi-definite. This process is stopped when the distance is deemed small enough. This final distance ϵ is called the *primal infeasibility*, and is one of the result quality measures displayed by SDP solvers[13]. Therefore, we do not obtain a Q satisfying $\mathrm{tr}(A_i Q) = a_i$ but rather $\mathrm{tr}(A_i Q) = a_i + \epsilon_i$ for some small ϵ_i such that $|\epsilon_i| \leq \epsilon$.

4.2 Proving Existence of a Nearby Solution

This primal infeasibility has a simple translation in terms of our original SOS problem. The polynomial equality $p = z^T Q\, z$ is encoded as one scalar constraint $\mathrm{tr}(A_i Q) = a_i$ for each coefficient a_i of the polynomial p (c.f., Examples 2). coefficients of the polynomials p and $z^T Q\, z$ differ by some ϵ_i and, since $|\epsilon_i| \leq \epsilon$, there exists a matrix $E \in \mathbb{R}^{s \times s}$ such that, for all i, j, $|E_{i,j}| \leq \epsilon$ and

$$p = z^T (Q + E)\, z. \tag{4}$$

Proving that $Q + E \succeq 0$ is now enough to prove that the polynomial p is SOS, hence non negative. A sufficient condition is to check[14] $Q - s\epsilon I \succeq 0$.

As seen in Section 3.2, checking that a matrix M is positive semi-definite amounts to exhibiting a matrix R such that $M = R^T R$. The Cholesky decomposition algorithm [45, §1.4] computes such a matrix R. Given a matrix $M \in \mathbb{R}^{s \times s}$, it attempts to compute R such that $M = R^T R$ and when M is not positive semi-definite, it fails by attempting to take the square root of a negative value or perform a division by zero.

Due to rounding errors, a simple floating-point Cholesky decomposition would produce a matrix R not exactly satisfying the equality $M = R^T R$, hence not proving $M \succeq 0$. However, these rounding errors can be bounded by a matrix B so that, when the floating-point Cholesky decomposition of $M - B$ succeeds, then $M \succeq 0$ is guaranteed to hold. Moreover, B can be easily computed from the matrix M and the characteristics of the floating-point format used [41].

To sum up, the following verification procedure can prove that a given polynomial p is SOS[15].

[13] Typically, $\epsilon \sim 10^{-8}$.

[14] In order to get good likelihood for this to hold, we ask the SDP solver for $Q - 2s\epsilon I \succeq 0$ rather than $Q \succeq 0$, as solvers often return matrices Q slightly not positive definite.

[15] It is worth noting that the value reported by the solver for ϵ, being just computed with floating-point arithmetic, cannot be formally trusted. It must then be recomputed.

Let $Q \in \mathbb{R}^{s \times s}$ be the approximate solution returned by an SDP solver for the problem $p = z^T Q\, z \wedge Q \succeq 0$. Then,

1. Compute a bound ϵ on the coefficients of $p - z^T Q\, z$.
2. Check that $Q - s\epsilon I \succeq 0$.

Complexity Note that step 1 can be achieved using floating-point interval arithmetic in $\Theta(s^2)$ operations while the Cholesky decomposition in step 2 requires $\Theta(s^3)$ floating-point operations. Thus, the whole verification method takes $\Theta(s^3)$ floating-point operations which, in practice, constitutes a very small overhead compared to the time required by the SDP solver to compute Q.

Soundness It is interesting to notice that the soundness of the method does not rely on the SDP solver. Thanks to this pessimistic method, the trusted code-base remains small, and efficient off-the-shelf solvers can be used as untrusted oracles. The method was even verified [31,38] within the Coq proof assistant.

Incompleteness Numerical verification methods can only prove inequalities satisfied with some margin. Here, if the polynomial p to prove SOS (hence $p \geq 0$) reaches the value 0, this usually means that the feasible set of the SDP problem $\left\{Q \;\middle|\; p = z^T Q\, z, Q \succeq 0\right\}$ has an empty relative interior (i.e., there is no point Q in this set such that a small ball centered on Q is included in $\{M \mid M \succeq 0\}$) and the method does not work, as illustrated on Figure 2. This is a second source of incompleteness of our approach, that adds to the limitation of degrees of polynomials searched for, as presented in Section 3.1.

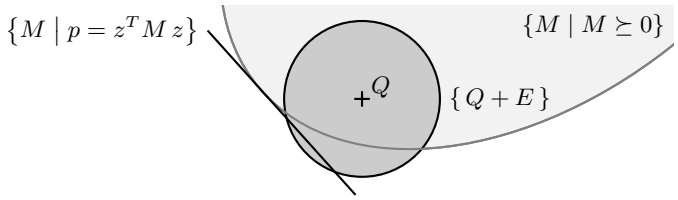

Fig. 2: When the feasible set has an empty interior, the subspace $\left\{M \;\middle|\; p = z^T M\, z\right\}$ is tangent to $\{M \mid M \succeq 0\}$. Thus the ball $\{\, Q + E \,\}$ intersecting the subspace almost never lies in $\{M \mid M \succeq 0\}$, making the proof fail.

Remark 3. The floating-point Cholesky decomposition is theoretically a third source of incompleteness. However, it is negligible as the entries of the bound matrix B are, in practice, orders of magnitude smallers than the accuracy ϵ of the SDP solvers [40].

4.3 Rounding to an Exact Rational Solution

The most common solution to verify results of SOS programming is to round the output of the SDP solver to an exact rational solution [19,24,33].

To sum up, the matrix Q returned by the SDP solver is first projected to the subspace $\left\{M \mid p = z^T M\, z\right\}$ then all its entries are rounded to rationals with small denominators (first integers, then multiples of $\frac{1}{2}, \frac{1}{3}, \ldots$)[16]. For each rounding, positive semi-definiteness of the resulting matrix Q is tested using a complete check, based on a LDLT decomposition[17] [19]. The rationale behind this choice is that problems involving only simple rational coefficients can reasonably be expected to admit simple rational solutions[18].

Using exact solutions potentially enables to verify SDP problems with empty relative interiors. This means the ability to prove inequalities without margin, to distinguish strict and non-strict inequalities and even to handle (dis)equalities. All of this nevertheless requires a different relaxation scheme than (1).

Example 4. To prove $x_1 \geq 0 \wedge x_2 \geq 0 \wedge q_1 = 0 \wedge q_2 = 0 \wedge p > 0$ unsatisfiable, with $q_1 := x_1^2 + x_2^2 - x_3^2 - x_4^2 - 2$, $q_2 := x_1x_3 + x_2x_4$ and $p := x_3x_4 - x_1x_2$, one can look for polynomials l_1, l_2 and SOS polynomials $s_1, \ldots, s_8$ such that $l_1q_1 + l_2q_2 + s_1 + s_2p + s_3x_1 + s_4x_1p + s_5x_2 + s_6x_2p + s_7x_1x_2 + s_8x_1x_2p + p = 0$.

Rounding the result of an SDP solver yields $l_1 = -\frac{1}{2}(x_1x_2 - x_3x_4)$, $l_2 = -\frac{1}{2}(x_2x_3 + x_1x_4)$, $s_2 = \frac{1}{2}\left(x_3^2 + x_4^2\right)$, $s_7 = \frac{1}{2}\left(x_1^2 + x_2^2 + x_3^2 + x_4^2\right)$ and $s_1 = s_3 = s_4 = s_5 = s_6 = s_8 = 0$. This problem has no margin, since when replacing $p > 0$ by $p \geq 0$, $(x_1, x_2, x_3, x_4) = (0, \sqrt{2}, 0, 0)$ becomes a solution.

Under some hypotheses, this relaxation scheme is complete, as stated by a theorem from Stengle [27, Th. 2.11]. However, similarly to Section 3.1, no practical bound is known on the degrees of the relaxation polynomials.

Complexity The relaxation scheme involves products of all polynomials appearing in the original problem constraints. The number of such products, being exponential in the number of constraints, limits the scalability of the approach.

Moreover, to actually enjoy the benefits of exact solutions, the floating-point Cholesky decomposition introduced in Section 4.2 cannot be used and has to be replaced by an exact rational decomposition[19]. Computing decompositions of large matrices can then become particularly costly as the size of the involved rationals can blow up exponentially during the computation.

Soundness The exact solutions make for an easy verification. The method is thus implemented in the HOL Light [19] and Coq [4] proof assistants.

[16] In practice, to ensure that the rounded matrix Q still satisfy the equality $p = z^T Q\, z$, a dual SDP encoding is used, that differs from the encoding introduced in Section 3. This dual encoding is also called image representation [36, §6.1].

[17] The LDLT decomposition expresses a positive semi-definite matrix M as $M = LDL^T$ with L a lower triangular matrix and D a diagonal matrix.

[18] However, there exist rational SDP problems that do not admit any rational solution.

[19] The Cholesky decomposition, involving square roots, cannot be computed in rational arithmetic, however its LDLT variant can.

Incompleteness Although this verification method can work for some SDP problems with an empty relative interior, the rounding heuristic is not guaranteed to provide a solution. In practice, it tends to fail on large problems or problems whose coefficients are not rationals with small numerators and denominators.

5 Experimental Results

5.1 The OSDP Library

The SOS to SDP translation described in Section 3, as well as the validation methods described in Section 4 have been implemented in our OCaml library OSDP. This library offers a common interface to the SDP solvers[20] Csdp [6], Mosek [2] and SDPA [46], giving simple access to SOS programming in contexts where soundness matters, such as SMT solvers or program static analyzers. It is composed of 5 kloc of OCaml and 1 kloc of C (interfaces with SDP solvers) and is available under LGPL license at `https://cavale.enseeiht.fr/osdp/`.

5.2 Integration of OSDP in Alt-Ergo

Alt-Ergo [5] is a very effective SMT solver for proving formulas generated by program verification frameworks. It is used as a back-end of different tools and in various settings, in particular via the Why3 [16] platform. For instance, the Frama-C [12] suite relies on it to prove formulas generated from C code, and the SPARK [21] toolset uses it to check formulas produced from Ada programs. It is also used by EasyCrypt [3] to prove formulas issued from cryptographic protocols verification, from the Cubicle [10] model-checker, and from Atelier-B [1].

Alt-Ergo's native input language is a polymorphic first-order logic *à la ML* modulo theories, a very suitable language for expressing formulas generated in the context of program verification. Its reasoning engine is built on top of a SAT solver that interacts with a combination of decision procedures to look for a model for the input formula. Universally quantified formulas, that naturally arise in program verification, are handled via E-matching techniques. Currently, Alt-Ergo implements decision procedures for the free theory of equality with uninterpreted symbols, linear arithmetic over integers and rationals, fragments of non-linear arithmetic, enumerated and records datatypes, and the theory of associative and commutative function symbols (hereafter AC).

Figure 3 shows the simplified architecture of arithmetic reasoning framework in Alt-Ergo, and the OSDP extension. The first component in the figure is a completion-like algorithm AC(LA) that reasons modulo associativity and commutativity properties of non-linear multiplication, as well as its distributivity over addition[21]. AC(LA) is a modular extension of ground AC completion with a decision procedure for reasoning modulo equalities of linear integer and rational

[20] Csdp is used for the following benchmarks as it provides the best results.
[21] Addition and multiplication by a constant is directly handled by the LA module.

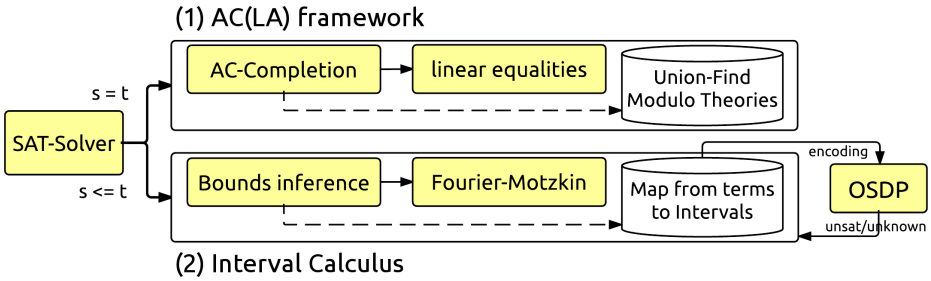

Fig. 3: Alt-Ergo's arithmetic reasoning framework with OSDP integration.

arithmetic [9]. It builds and maintains a convergent term-rewriting system modulo arithmetic equalities and the AC properties of the non-linear multiplication symbol. The rewriting system is used to update a union-find data-structure.

The second component is an Interval Calculus algorithm that computes bounds of (non-linear) terms: the initial non-linear problem is first relaxed by abstracting non-linear parts, and a Fourier-Motzkin extension[22] is used to infer bounds on the abstracted linear problem. In a second step, axioms of non-linear arithmetic are internally applied by intervals propagation. These two steps allow to maintain a map associating the terms of the problems (that are normalized *w.r.t.* the union-find) to unions of intervals.

Finally, the last part is the SAT solver that dispatches equalities and inequalities to the right component and performs case-split analysis over finite domains. Of course, this presentation is very simplified and the exact architecture of Alt-Ergo is much more complicated.

The integration of OSDP in Alt-Ergo is achieved via the extension of the Interval Calculus component of the solver, as shown in Figure 3: terms that are polynomials, and their corresponding interval bounds, form the problem (1) which is given to OSDP. OSDP attempts to verify its result with the method of Section 4.2. When it succeeds, the original conjunction of constraints is proved `unsat`. Otherwise, (dis)equalities are added and OSDP attempts a new proof by the method of Section 4.3. In case of success, `unsat` is proved, otherwise satisfiability or unsatisfiability cannot be deduced. Outlines of the first algorithm are given in Figure 4 whereas the second one follows the original implementation [19].

Our modified version of Alt-Ergo is available under CeCILL-C license at `https://cavale.enseeiht.fr/osdp/aesdp/`.

Incrementality In the SMT context, our theory solver is often succesively called with the same problem with a few additional constraints each time. It would then be interesting to avoid doing the whole computation again when a constraint is added, as is usually done with the simplex algorithm for linear arithmetic.

Some SDP solvers do offer to provide an initial point. Our experiments however indicated that this significantly speeds up the computation only when

[22] We can also use a simplex-based algorithm [8] for bounds inference.

$p'_1 := (p_1 - a_1)(b_1 - p_1)$, ..., $p'_k := (p_k - a_k)(b_k - p_k)$
// or $p'_i := p_i - a_i$ when $b_i = +\infty$ or $p'_i := b_i - p_i$ when $a_i = -\infty$
$d := \max_i \{ \deg(p'_i) \}$
encode $-\sum_{i=1}^{k} r_i p'_i$ is SOS, r_1 is SOS, ... r'_k is SOS
as an SDP problem $-\sum r_i p'_i = z_0^T Q_0 z_0$, $r_1 = z_1^T Q_1 z_1$, ..., $r_k = z_k^T Q_k z_k$
with $\deg(r_i) := 2 \left\lceil \frac{d - \deg(p'_i)}{2} \right\rceil$
call an SDP solver and retrieve r_1, r_k and Q_0, Q_1, ..., Q_k
overapproximate $\epsilon_i := \max_\alpha \left\{ |c_\alpha| \;\middle|\; r_i - z_i^T Q_1 z_i = \sum_\alpha c_\alpha x^\alpha \right\}$
if $1 \in z_0 \wedge Q_0 - \#|z_0|\epsilon_0 I \succ 0 \wedge Q_1 - \#|z_1|\epsilon_1 I \succ 0 \wedge \ldots \wedge Q_k - \#|z_k|\epsilon_k I \succ 0$ **then**
  return Unsat
else
  return Unknown
end if

Fig. 4: Semi-decision procedure to prove $\bigwedge_{i=1}^{k} p_i \in [a_i, b_i]$ unsat. $\#|z|$ is the size of the vector z and $\succ 0$ is tested with a floating-point Cholesky decomposition [41].

the provided point is extremely close to the solution. A bad initial point could even slow down the computation or, worse, make it fail. This is due to the very different nature of the interior point algorithms, compared to the simplex, and their convergence properties [7, Part III]. Thus, speed ups could only be obtained when the previous set of constraints was already unsatisfiable, i.e. a useless case.

Small Conflict Sets When a set of constraints is unsatisfiable, some of them may not play any role in this unsatisfiability. Returning a small subset of unsatisfiable constraints can help the underlying SAT solver. Such useless constraints can easily be identified in (1) when the relaxation polynomial r_i is 0. A common heuristic to maximize their number is to ask the SDP solver to minimize (the sum of) the traces of the matrices Q_i.

When using the exact method of Section 4.3, the appropriate r_i are exactly 0. Things are not so clear when using the approximate method of Section 4.2 since the r_i are only *close to* 0. A simple solution is to rank the r_i by decreasing trace of Q_i before performing a dichotomy search for the smallest prefix of this sequence proved unsatisfiable. Thus, for n constraints, $\log(n)$ SDPs are solved.

5.3 Experimental Results

We compared our modified version of Alt-Ergo (v. 1.30) to the SMT solvers ran in both the QF_NIA and QF_NRA sections of the last SMT-COMP. We ran the solvers on two sets of benchmarks. The first set comes from the QF_NIA and QF_NRA benchmarks for the last SMT-COMP. The second set contains

four subsets. The `C` problems are generated by Frama-C/Why3 [12,16] from control-command C programs such as the one from Section 2, with up to a dozen variables [11,39]. To distinguish difficulties coming from the handling of the memory model of C, for which Alt-Ergo was particularly designed, and from the actual non-linear arithmetic problem, the `quadratic` benchmarks contain simplified versions of the `C` problems with a purely arithmetic goal. To demonstrate that the interest of our approach is not limited to this initial target application, the `flyspeck` benchmarks come from the benchmark sets of dReal[23] [18] and `global-opt` are global optimization benchmarks [34]. All these benchmarks are available at `https://cavale.enseeiht.fr/osdp/aesdp/`. Since our solver only targets unsat proofs, benchmarks known sat were removed from both sets.

Table 1: Experimental results on benchmarks from QF_NIA.

		AE	AESDP	AESDPap	AESDPex	CVC4	Smtrat	Yices2	Z3
AProVE (746)	unsat	103	319	359	318	586	185	**709**	252
	time	7387	23968	7664	22701	10821	3879	**1982**	5156
calypto (97)	unsat	92	88	88	89	87	89	**97**	95
	time	357	679	489	816	7	754	**409**	613
LassoRanker (102)	unsat	57	62	64	63	72	20	**84**	84
	time	9	959	274	878	27	12	**595**	2538
LCTES (2)	unsat	0	0	0	0	**1**	0	0	0
	time	0	0	0	0	**0**	0	0	0
leipzig (5)	unsat	0	0	0	0	0	0	**1**	0
	time	0	0	0	0	0	0	**0**	0
mcm (161)	unsat	0	0	0	0	**4**	0	0	4
	time	0	0	0	0	**2489**	0	0	2527
UltimateAutom (7)	unsat	1	7	7	7	6	1	**7**	7
	time	0.35	0.73	0.62	0.69	0.03	7.22	**0.04**	0.31
UltimateLasso (26)	unsat	26	26	26	26	4	26	**26**	26
	time	118	212	126	215	66	177	**6**	21
total (1146)	unsat	279	502	544	503	780	321	**924**	468
	time	7872	25818	8553	24611	13411	4829	**2993**	10855

All experiments were conducted on an Intel Xeon 2.30 GHz processor, with individual runs limited to 2GB of memory and 900 seconds. The results are presented in Tables 1, 2 and 3. For each subset of problems, the first column indicates the number of problems that each solver managed to prove unsat and the second presents the cumulative time (in seconds) for these problems.AE is the original Alt-Ergo, AESDP our new version, AESDPap the same but using only the approximate method of Section 4.2 and AESDPex using only the exact method of Section 4.3. All solvers were run with default options, except CVC4 which was run with all its `--nl-ext*` options.

[23] Removing problems containing functions sin and cos, not handled by our tool.

Table 2: Experimental results on benchmarks from QF_NRA.

		AE	AESDP	AESDPap	AESDPex	CVC4	Smtrat	Yices2	Z3
Sturm-MBO (300)	unsat	155	155	155	155	285	**285**	2	47
	time	12950	13075	13053	12973	1403	**620**	0	21
Sturm-MGC (7)	unsat	0	0	0	0	1	1	0	**7**
	time	0	0	0	0	7	0	0	**0**
Heizmann (68)	unsat	0	0	0	0	1	1	**11**	3
	time	0	0	0	0	16	0	**2083**	41
hong (20)	unsat	1	20	20	20	20	**20**	8	9
	time	0	28	24	27	1	**0**	240	6
hycomp (2494)	unsat	1285	1266	1271	1265	2184	1588	2182	**2201**
	time	15351	15857	16080	14909	208	13784	1241	**4498**
keymaera (320)	unsat	261	291	278	291	249	307	270	**318**
	time	36	356	97	360	4	13	359	**2**
LassoRanker (627)	unsat	0	0	0	0	**441**	0	236	119
	time	0	0	0	0	**32786**	0	30835	1733
meti-tarski (2615)	unsat	1882	2273	2267	2241	1643	2520	2578	**2611**
	time	10	91	65	73	804	3345	2027	**337**
UltimateAutom (13)	unsat	0	0	0	0	5	0	12	**13**
	time	0	0	0	0	0.52	0	57.19	**19.23**
zankl (85)	unsat	14	24	24	24	24	19	**32**	27
	time	1.00	15.46	16.09	15.67	9.40	13.47	**7.22**	0.43
total (6549)	unsat	3571	4029	4015	3996	4853	4740	5331	**5355**
	time	28348	29423	29334	28357	35239	17775	36849	**6658**

As seen in Table 1 and 2, despite an improvement over Alt-Ergo alone, our development is not competitive with state-of-the-art solvers on the QF_NIA and QF_NRA benchmarks. In fact, the set of problems solved by any of our Alt-Ergo versions is strictly included in the set of problems solved by at least one of the other solvers. The most commonly observed source of failure for AESDPap here comes from SDPs with empty relative interior. Although AESDPex can handle such problems, it is impaired by its much higher complexity.

However good results are obtained on the more numerical[24] second set of benchmarks. In particular, control-command programs with up to a dozen variables are verified while other solvers remain limited to two variables. Playing a key point in this result, the inequalities in these benchmarks are satisfied with some margin. For control command programs, this comes from the fact that they are designed to be robust to many small errors. This opens new perspectives for the verification of functional properties of control-command programs, particularly in the aerospace domain, our main application field at ONERA[25].

Although solvers such as dReal, based on branch and bound with interval arithmetic could be expected to perform well on these numerical benchmarks,

[24] Involving polynomials with a few dozen monomials or more and whose coefficients are not integers or rationals with small numerators and denominators.

[25] French public agency for aerospace research.

Table 3: Experimental results on benchmarks from [11,18,34,39].

		AE	AESDP	AESDPap	AESDPex	CVC4	Smtrat	Yices2	Z3	dReal
C (67)	unsat	11	**63**	63	13	0	0	0	0	0
	time	0.05	**39.78**	40.01	1.18	0	0	0	0	0
quadratic (67)	unsat	13	**67**	67	15	14	18	25	25	13
	time	0.06	**14.68**	15.44	0.08	2.46	1.26	357.20	257.39	23.36
flyspeck (20)	unsat	1	**19**	19	3	6	9	10	9	16
	time	0.00	**26.35**	26.62	0.01	695.59	36.54	0.05	0.05	11.77
global-opt (14)	unsat	2	**14**	14	5	5	12	12	13	9
	time	0.01	**8.72**	8.83	0.20	0.12	41.18	0.16	683.45	0.05
total (168)	unsat	27	**163**	163	36	25	39	47	47	38
	time	0.12	**89.53**	90.90	1.47	698.17	78.98	357.41	940.89	35.18

dReal solves less benchmarks than most other solvers. Geometrically speaking, the `C` benchmarks require to prove that an ellipsoid is included in a slightly larger one, i.e., the borders of both ellipsoids are close from one another. This requires to subdivide the space between the two borders in many small boxes so that none of them intersects both the interior of the first ellipsoid and the exterior of the second one. Whereas this can remain tractable for small dimensional ellipsoids, the number of required boxes grows exponentially with the dimension, which explains the poor results of dReal. This issue is unfortunately shared, to a large extent, by any linear relaxation, including more elaborate ones [30].

6 Related Work and Conclusion

Related work. Monniaux and Corbineau [33] improved the rounding heuristic of Harrison [19]. This has unfortunately no impact on the complexity of the relaxation scheme. Platzer et al. [37] compared their early versions with the symbolic methods based on quantifier elimination and Gröbner basis. An intermediate solution is offered by Magron et al. [29] but only handling a restricted class of parametric problems.

Branch-and-bound and interval arithmetic constitute another numerical approach to non-linear arithmetic, as implemented in the SMT solver dReal by Gao et al. [17,18]. These methods easily handle non-linear functions such as the trigonometric functions sin or cos, not yet considered in our prototype[26]. In the case of polynomial inequalities Muñoz and Narkawicz [34] offer Bernstein polynomials as an improvement to simple interval arithmetic.

Finally, VSDP [20,22] is a wrapper to SDP solvers offering a similar method to the one of Section 4.2. Moreover, an implementation is also offered by Löfberg [28] in the popular Matlab interface Yalmip but remains unsound, since all computations are performed with floating-point arithmetic, ignoring rounding errors.

[26] Polynomial approximations such as Taylor expansions should be investigated.

Using convex optimization into an SMT solver was already proposed by Nuzzo et al. [35,43]. However, they intentionally made their solver unsound in order to lean toward completeness. While this can make sense in a bounded model checking context, soundness is required for many applications, such as program verification. Moreover, this proposal was limited to convex formulas. Although this enables to provide models for satisfiable formulas, while only unsat formulas are considered in this paper, and whereas this seems a perfect choice for bounded model checking applications, non convex formulas are pervasive in applications such as program verification[27].

The use of numerical off-the-shelf solvers in SMT tools has also been studied in the framework of linear arithmetic [15,32]. Some comparison with state-of-the-art exact simplex procedures show mitigated results [14] but better results can be obtained by combining both approaches [25].

Conclusion. We presented a semi-decision procedure for non-linear polynomial constraints over the reals, based on numerical optimization solvers. Since these solvers only compute approximate solutions, a-posteriori soundness checks were investigated. Our first prototype implemented in the Alt-Ergo SMT solver shows that, although the new numerical method does not strictly outperform state-of-the-art symbolic methods, it enables to solve practical problems that are out of reach for other methods. In particular, this is demonstrated on the verification of functional properties of control-command programs. Such properties are of significant importance for critical cyber-physical systems.

It could thus be worth studying the combination of symbolic and numerical methods in the hope to benefit from the best of both worlds.

Data Availability Statement and Acknowledgements. The source code, benchmarks and instructions to replicate the results of Section 5 are available in the figshare repository: `https://doi.org/10.6084/m9.figshare.5900260`

The authors thank Rémi Delmas for insightful discussions and technical help, particularly with the dReal solver.

References

1. Jean-Raymond Abrial. *The B-book - assigning programs to meanings.* Cambridge University Press, 2005.
2. MOSEK ApS. *The MOSEK C optimizer API manual Version 7.1 (Rev. 40)*, 2015.
3. Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. Easycrypt: A tutorial. In *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*, 2013.
4. Frédéric Besson. Fast reflexive arithmetic tactics the linear case and beyond. In *TYPES*, 2006.

[27] Typically, to prove a convex loop invariant I for a loop body f, one need to prove $I \Rightarrow I(f)$, that is $\neg I \vee I(f)$ which is likely non convex ($\neg I$ being concave).

5. François Bobot, Sylvain Conchon, Evelyne Contejean, Mohamed Iguernlala, Stephane Lescuyer, and Alain Mebsout. *Alt-Ergo, version 0.99.1*. CNRS, Inria, Université Paris-Sud 11, and OCamlPro, December 2014. `http://alt-ergo.lri.fr/`.
6. Brian Borchers. CSDP, A C library for semidefinite programming. *Optimization Methods and Software*, 1999.
7. Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
8. François Bobot, Sylvain Conchon, Evelyne Contejean, Mohamed Iguernelala, Assia Mahboubi, Alain Mebsout, and Guillaume Melquiond. A Simplex-Based Extension of Fourier-Motzkin for Solving Linear Integer Arithmetic. In *IJCAR*, 2012.
9. Sylvain Conchon, Évelyne Contejean, and Mohamed Iguernelala. Canonized rewriting and ground AC completion modulo Shostak theories : Design and implementation. *Logical Methods in Computer Science*, 2012. Selected Papers of TACAS.
10. Sylvain Conchon, Amit Goel, Sava Krstic, Alain Mebsout, and Fatiha Zaïdi. Cubicle: A Parallel SMT-Based Model Checker for Parameterized Systems. In *CAV*, 2012.
11. Arlen Cox, Sriram Sankaranarayanan, and Bor-Yuh Evan Chang. A Bit Too Precise? Bounded Verification of Quantized Digital Filters. In *TACAS*, 2012.
12. Pascal Cuoq, Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski. Frama-C - a software analysis perspective. In *SEFM*, 2012.
13. Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *TACAS*, 2008.
14. Diego Caminha B de Oliveira and David Monniaux. Experiments on the feasibility of using a floating-point simplex in an SMT solver. In *PAAR@ IJCAR*, 2012.
15. Germain Faure, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. SAT modulo the theory of linear arithmetic: Exact, inexact and commercial solvers. In *SAT*, 2008.
16. Jean-Christophe Filliâtre and Andrei Paskevich. Why3 - Where Programs Meet Provers. In *ESOP*, 2013.
17. Sicun Gao, Jeremy Avigad, and Edmund M. Clarke. δ-complete decision procedures for satisfiability over the reals. In *IJCAR*, 2012.
18. Sicun Gao, Soonho Kong, and Edmund M. Clarke. dreal: An SMT solver for nonlinear theories over the reals. In *CADE*, 2013.
19. John Harrison. Verifying nonlinear real formulas via sums of squares. In *TPHOLs*, 2007.
20. Viktor Härter, Christian Jansson, and Marko Lange. VSDP: verified semidefinite programming. `http://www.ti3.tuhh.de/jansson/vsdp/`.
21. Duc Hoang, Yannick Moy, Angela Wallenburg, and Roderick Chapman. SPARK 2014 and gnatprove - A competition report from builders of an industrial-strength verifying compiler. *STTT*, 2015.
22. Christian Jansson, Denis Chaykin, and Christian Keil. Rigorous error bounds for the optimal value in semidefinite programming. *SIAM J. Numerical Analysis*, 2007.
23. Dejan Jovanovic and Leonardo Mendonça de Moura. Solving non-linear arithmetic. In *IJCAR*, 2012.
24. Erich Kaltofen, Bin Li, Zhengfeng Yang, and Lihong Zhi. Exact certification in global polynomial optimization via sums-of-squares of rational functions with rational coefficients. *J. Symb. Comput.*, 2012.
25. Tim King, Clark W. Barrett, and Cesare Tinelli. Leveraging linear and mixed integer programming for SMT. In *FMCAD*, 2014.
26. Jean Bernard Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 2001.

27. Jean Bernard Lasserre. *Moments, Positive Polynomials and Their Applications*. Imperial College Press Optimization. Imperial College Press, 2009.
28. Johan Löfberg. Pre- and post-processing sum-of-squares programs in practice. *IEEE Transactions on Automatic Control*, 2009.
29. Victor Magron, Xavier Allamigeon, Stéphane Gaubert, and Benjamin Werner. Formal proofs for nonlinear optimization. *Journal of Formalized Reasoning*, 2015.
30. Alexandre Maréchal, Alexis Fouilhé, Tim King, David Monniaux, and Michaël Périn. Polyhedral approximation of multivariate polynomials using handelman's theorem. In *VMCAI*, 2016.
31. Érik Martin-Dorel and Pierre Roux. A reflexive tactic for polynomial positivity using numerical solvers and floating-point computations. In *CPP*, 2017.
32. David Monniaux. On using floating-point computations to help an exact linear arithmetic decision procedure. In *CAV*, 2009.
33. David Monniaux and Pierre Corbineau. On the generation of positivstellensatz witnesses in degenerate cases. In *ITP*, 2011.
34. César A. Muñoz and Anthony Narkawicz. Formalization of Bernstein Polynomials and Applications to Global Optimization. *J. Autom. Reasoning*, 2013.
35. Pierluigi Nuzzo, Alberto Puggelli, Sanjit A. Seshia, and Alberto L. Sangiovanni-Vincentelli. Calcs: SMT solving for non-linear convex constraints. In *FMCAD*, 2010.
36. Pablo A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Math. Program.*, 2003.
37. André Platzer, Jan-David Quesel, and Philipp Rümmer. Real world verification. In *CADE*, 2009.
38. Pierre Roux. Formal proofs of rounding error bounds - with application to an automatic positive definiteness check. *J. Autom. Reasoning*, 2016.
39. Pierre Roux, Romain Jobredeaux, Pierre-Loïc Garoche, and Éric Féron. A generic ellipsoid abstract domain for linear time invariant systems. In *HSCC*, 2012.
40. Pierre Roux, Yuen-Lam Voronin, and Sriram Sankaranarayanan. Validating numerical semidefinite programming solvers for polynomial invariants. In *SAS*, 2016.
41. Siegfried M. Rump. Verification of positive definiteness. *BIT Num. Math.*, 2006.
42. Siegfried M. Rump. Verification methods: Rigorous results using floating-point arithmetic. *Acta Numerica*, 2010.
43. Yasser Shoukry, Pierluigi Nuzzo, Alberto L. Sangiovanni-Vincentelli, Sanjit A. Seshia, George J. Pappas, and Paulo Tabuada. SMC: satisfiability modulo convex optimization. In *HSCC*, 2017.
44. Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM Review*, 1996.
45. David S Watkins. *Fundamentals of matrix computations*. John Wiley & Sons, 2004.
46. Makoto Yamashita, Katsuki Fujisawa, Kazuhide Nakata, Maho Nakata, Mituhiro Fukuda, Kazuhiro Kobayashi, and Kazushige Goto. A high-performance software package for semidefinite programs: Sdpa 7. Technical Report B-460, Tokyo Institute of Technology, Tokyo, 2010.